



Universidad Nacional de Rosario
Facultad de Ciencias Exactas,
Ingeniería y Agrimensura
Departamento de Física – ECEN



FÍSICA COMPUTACIONAL

Desarrollo del software PyReduc para reducción de imágenes astronómicas

Carlos Mauricio Silva

16 de diciembre de 2018

Resumen

En este trabajo se presenta el desarrollo del software PyReduc, cuyo objetivo es reducir y apilar imágenes astronómicas en formato FITS. Se comentan los fundamentos de la reducción de imágenes y se explican las funciones principales del programa.

1. Introducción teórica

1.1. Motivación

Si hay una ciencia en la que cualquier ciudadano sin formación académica de grado puede hacer una contribución con un mínimo de equipamiento, esta es la Astronomía. Con la popularización de las cámaras “DSLR” (cámara digital réflex de una sola lente), cualquier persona con un poco de tiempo puede aprender la técnica de astrometría y fotometría. Organizaciones como la Unión Astronómica Internacional (IAU) o la Asociación Americana de Observadores de Estrellas Variables (AAVSO) reciben las contribuciones de los observadores amateurs de todo el mundo, a quienes a través de diversas campañas se les ha comenzado a llamar “científicos ciudadanos” [1].

Si bien en Internet se puede encontrar software gratuito y de alta calidad para realizar toda la tarea de procesado de imágenes astronómicas, estos están pensados principalmente para procesamientos artísticos o no están lo suficientemente automatizados para cumplir todas las tareas de preprocesado requeridas para poder reportar información con valor científico.

El objetivo de este trabajo es describir el funcionamiento del software **PyReduc**, desarrollado bajo las siguientes directivas:

- Cumplir con las libertades del Software Libre [2].
- Constituir un conjunto de subrutinas apto para realizar la reducción de imágenes, es decir el preprocesado necesario para obtener imágenes listas para realizar mediciones fotométricas sobre ellas.
- Que el proceso de reducción sea, en la medida de lo posible, automático.
- Que maneje el formato FITS, un formato de archivos estándar en investigación astronómica a nivel mundial.
- Que esté programado en un lenguaje de programación moderno, bien documentado y con librerías científicas de comprobada utilidad, como lo es el lenguaje Python.

1.2. Algunos conceptos: Fotometría DSLR

El objetivo de la fotometría es medir la intensidad de la luz que nos llega de los objetos celestes. Por ejemplo, en el caso de una estrella variable, esta información se hace relevante cuando se quiere obtener información sobre la evolución de la estrella en estudio.

El término “DSLR” se refiere a una clase genérica de cámaras que es adecuada para llevar a cabo observaciones fotométricas. Recientemente, muchas cámaras automáticas han comenzado a incorporar varias características requeridas para fotometría astronómica [1].

En las cámaras DSLR, la imagen es capturada por un sensor electrónico de tipo CMOS (Semiconductor Complementario de Oxido Metálico), sobre el que se dispone un mosaico que filtra la luz en tres colores: rojo (R), verde (G) y azul (B). Estos mosaicos se conocen como mosaicos de Bayer. Dependiendo del modelo y marca comercial de la cámara, estos mosaicos se pueden disponer en distintos órdenes. Así, por ejemplo, una la mayoría de las cámaras DSLR marca **Canon** llevan un mosaico RGGB, lo que quiere decir que contiene dos celdas que detectan el verde, por cada celda de rojo y cada celda de azul. La estructura de estas cámaras escapa al objetivo de este trabajo, pero puede consultarse en la bibliografía [1].

El sensor en sí mismo está hecho de un chip de silicio sobre el cual está grabado el circuito CMOS. El elemento sensible a los fotones en cada píxel es un fotodiodo (o un sensor fotoeléctrico MOS). Tomemos, para orientarnos mejor, un único pixel en el sensor. Durante la exposición fotográfica, los fotones que van llegando al fotodiodo excitan electrones, los cuales se van almacenando en un capacitor originalmente descargado. Al final de la exposición, se lee la tensión en bornes del capacitor. Estas tensiones son muy pequeñas, por lo que para realizar la lectura se amplifican electrónicamente y luego un conversor analógico-digital (ADC) se encarga de convertirla en un número binario. En la fotometría DSLR se suele decir que este número está medido en ADU (Unidad Analógica-Digital). Este valor en ADU es el que se almacena en la matriz que conforma el archivo resultante.

Es interesante mencionar que, dado que los valores ADU registrados en la imagen son proporcionales a la cantidad de fotones que llegaron a cada pixel en cuestión, es un valor adecuado para tomar a la hora de hacer fotometría.

1.3. El formato FITS

Llamaremos a las imágenes de astros capturadas con fines científicos “**tomas científicas**” o “**lights**”. Las cámaras DSLR comerciales traen múltiples funciones. Entre ellas, hay algoritmos de balance de blancos, reducción de ruido, compensación de exposición, etcétera. La mayoría de estos algoritmos operan sobre los datos que capta el sensor y los modifican, por lo que es fundamental *no usarlos* para la fotometría. Todas las mejoras sobre las imágenes se deben hacer en perfecto control por parte del usuario. Es por ello que, al captar una toma científica, no se deben guardar en formato JPG. Al guardar en formato JPG, la imagen se convierte a un espacio de color llamado sRGB que es no lineal, y por lo tanto, no fotométrico. El formato para almacenar tomas científicas sin pérdida de información es llamado RAW. Un archivo RAW está formado, básicamente, por una matriz de números binarios. Cada elemento de la matriz contiene el valor ADU de cada pixel de la imagen. Además los archivos RAW conservan, en los *metadatos* de la imagen, información sobre la captura: marca y modelo de la cámara, sensibilidad ISO, velocidad de obturación, datos de la lente utilizada, si estuviera disponible, etcétera.

Dada la gran variedad de sensores CMOS y matrices Bayer, cada marca y modelo de cámara tiene su propio formato RAW. Para las diversas operaciones de medición astronómicas, se hace necesario tener un formato universal de imágenes. Es así que en el campo de la Astronomía y la Astrofísica se ha hecho extensivo el uso del formato FITS.

FITS es el acrónimo de *Flexible Image Transport System* (Sistema Flexible de Transporte de Imágenes). El formato FITS se empezó a desarrollar a fines de la década de 1970, con el uso de las primeras cámaras CCD, a los fines de compartir datos astronómicos entre instituciones que cooperaban entre sí. Una reseña histórica puede leerse en la bibliografía [3]. Si bien la “T” en FITS significa “Imágenes”, hoy en día sería más adecuada la palabra “información”, ya que el formato FITS es mucho más que un formato de imagen. Como se establece en el sitio web de la Oficina de Soporte FITS de la NASA [6], en un archivo FITS se pueden guardar:

- Arrays multidimensionales: Espectros 1D, imágenes 2D, Cubos de datos de 3 o más dimensiones.
- Tablas que contengan filas y columnas de información (por ejemplo, coordenadas astrométricas).

- Una Cabecera con palabras clave (Header keywords) que provean información descriptiva sobre los datos.

Un archivo FITS se divide, esencialmente en dos partes: La cabecera (header) y los datos (data).

```
-----  
SIMPLE = T / file does conform to FITS standard  
BITPIX = 16 / number of bits per data pixel  
NAXIS = 3 / number of data axes  
NAXIS1 = 5208 / length of data axis 1  
NAXIS2 = 3476 / length of data axis 2  
NAXIS3 = 3 / length of data axis 3  
EXTEND = T / FITS dataset may contain extensions  
COMMENT FITS (Flexible Image Transport System) format is defined in 'Astronomy  
COMMENT and Astrophysics', volume 376, page 359; bibcode: 2001A&A...376..359H  
BZERO = 32768. / offset data range to that of unsigned short  
BSCALE = 1. / default scaling factor  
INSTRUME= 'Canon EOS 700D' / instrument name  
DATE = '2018-08-28T03:36:07' / UTC date that FITS file was created  
DATE-OBS= '2018-04-14T01:36:33' / YYYY-MM-DDThh:mm:ss observation start, UT  
XBINNING= 1 / Camera binning mode  
YBINNING= 1 / Camera binning mode  
EXPTIME = 15. / Exposure time [s]  
ISOSPEED= 1600. / ISO camera setting  
BAYERPAT= 'RGGB' , / Bayer color pattern  
PROGRAM = 'Siril v0.9.9' / Software that created this HDU  
-----
```

Figura 1: Ejemplo de cabecera de un archivo FITS. Las palabras en mayúscula a la izquierda del signo = son las palabras clave de la cabecera. El signo / indica un comentario.

Cabecera FITS (Header) Es una sección del archivo FITS que contiene palabras clave que dan información sobre el archivo. Cuando se convierte una imagen RAW a un archivo FITS, es habitual que los *metadatos* de la imagen se guarden en la cabecera. También durante el procesado, esta información se va ampliando. Por tradición, la cabecera de los archivos FITS está codificada en caracteres de 7 bits ASCII.

Un ejemplo de cabecera FITS se muestra en la Figura 1.

No se presentará en este trabajo una descripción exhaustiva de las palabras claves obligatorias y opcionales que puede tener la cabecera de una archivo FITS. En cambio, se comentarán las más relevantes a los fines de este trabajo. Para más detalles, se puede consultar el manual del Estándar FITS de la Unión Astronómica Internacional [7].

1. **BITPIX:** Indica el número de bits por pixel (o elemento de matriz) en la sección de datos.
2. **NAXIS:** Cantidad de ejes de datos. Se puede pensar como la dimensión de la matriz o el cubo de datos. En el ejemplo de la Figura 1, **NAXIS1** y **NAXIS2** corresponden a los dos

ejes de la imagen, mientras que **NAXIS3** corresponde a la cantidad de capas de color. Así podríamos decir que la imagen referenciada en el ejemplo tiene 5208×3476 pixeles y tres capas de color. La imagen de este ejemplo ya fue separada en los tres colores de los filtros del sensor. Las imágenes RAW que provienen de cámaras DSLR, deben ser separadas en colores de acuerdo con la organización del mosaico Bayer. Este proceso de descomposición se conoce como **debayerizado** (debayering) o **desmosaicado** (demosaicing). Si **NAXIS** vale n , entonces deberán estar, necesariamente, las palabras claves **NAXIS1**, **NAXIS2**, ..., **NAXISn**.

3. **INSTRUME**: Indica marca y modelo del instrumento utilizado. Se obtiene, habitualmente, de los metadatos de la imagen RAW.
4. **BAYERPAT**: Indica el patrón del mosaico Bayer de la cámara. No suele estar en los metadatos de la imagen RAW, así que al convertir el archivo a FITS se suministra esa información desde una base de datos de modelos de cámaras.
5. **EXPTIME**: Es el tiempo de exposición de la imagen, medido en segundos. Equivale al tiempo que el obturador permanece abierto en una cámara convencional. En fotografía astronómica artística estos tiempos de exposición suelen ser de varias horas. A los fines de la fotometría, no es necesario tanto tiempo de exposición, fundamentalmente si se tiene en cuenta que la cámara tiene una respuesta lineal en un cierto rango de ADU. Cuando los tiempos de exposición son muy largos, puede suceder que el objeto de interés esté *sobreexpuesto*. En ese caso, el valor ADU registrado no será un valor proporcional a la cantidad de fotones captados por el sensor. Luego, para elegir el tiempo de exposición, un factor clave a tener en cuenta es la *curva de linealidad* del sensor.
6. **ISOSPEED**: Es la configuración ISO en la que fue tomada la imagen. Muchas veces se dice que la configuración ISO se relaciona con la “sensibilidad” del sensor, pero en realidad la sensibilidad no cambia al cambiar el ajuste ISO. Lo que verdaderamente cambia con esta configuración es la **ganancia** del sensor. La ganancia se suele medir en electrones por ADU (e^-/ADU), y es una medida de la amplificación de la señal. Cuando un fotón llega al sensor, existe siempre la misma probabilidad de que este excite un electrón produciendo una señal. Esta respuesta a la excitación sería la sensibilidad del sensor. La ganancia, en cambio, está caracterizada por la amplificación de la señal producida. Configuraciones ISO más altas dan una ganancia más pequeña. Esto significa que para obtener una unidad ADU, a ISO más alto, se necesitará excitar menos electrones. Resumiendo, podemos decir que a mayor ISO, mayor amplificación de la señal, y menor ganancia. Un problema que surge al aumentar la configuración ISO es que también aumenta el ruido. Es por ello que, para fotografía astronómica no se usan las configuraciones ISO más altas. En fotografía astronómica artística, las configuraciones ISO suelen estar entre ISO 800 y 3200. En cambio, para las técnicas de fotometría, se busca que la ganancia del sensor sea cercana a $1 e^-/ADU$. Como ejemplo, en la DSLR **Canon EOS 700D**, esto se logra para ISO 200 o ISO 400.
7. **DATE**: Es la fecha y hora de creación del archivo FITS.
8. **DATE-OBS**: Es la fecha y hora en que se realizó la tomafotográfica, en *Tiempo Universal Coordinado*. Este valor se obtiene de los metadatos de la imagen, que a su vez se obtienen

de la configuración de la cámara. Para esto, es importante que la hora de la cámara esté bien sincronizada con la hora oficial.

Como decíamos, otras palabras claves son posibles, como la ubicación geográfica del observatorio, el nombre del observador, el nombre del objeto observado, sus coordenadas celestes, etcétera.

Datos FITS Son los datos propiamente dichos, codificados en binario según la palabra clave **BITPIX**, y ordenados en un arreglo de dimensiones indicadas por las palabras claves **NAXIS1**, **NAXIS2**, ..., **NAXISn** de la cabecera. Con estos datos se trabajará en la reducción de imágenes y la fotometría.

1.4. Reducción de imágenes: Tomas Dark, Bias y Flat

El proceso de reducción implica la eliminación de efectos instrumentales que estén presentes en los datos, ya sean de espectroscopía o de imagen directa. Es necesario llevar a cabo la reducción antes de poder realizar cualquier tipo de medida sobre nuestros datos. En este trabajo nos limitaremos a los efectos comúnmente presentes en las imágenes DSLR, pero esencialmente estos son los mismos efectos que se presentan en las imágenes con sensores CCD.

Podríamos decir que el mayor problema con el que hay que lidiar en la fotometría DSLR es con el ruido. Dado que en la fotometría necesitamos relevar la intensidad de la luz que proviene de las estrellas, necesitamos que el ruido sea lo suficientemente bajo para que no interfiera significativamente con las mediciones. Es por ello que el objetivo principal que se persigue con el proceso de reducción es incrementar la “**relación señal/ruido**” (SNR).

1.4.1. Tomas Bias

Al igual que en las tomas Dark, las tomas Bias se realizan con el objetivo tapado, y la misma configuración ISO que las tomas científicas, pero con la exposición más corta que proporcione la cámara (por ejemplo, 1/4000 s). Estas tomas contienen el ruido electrónico y de lectura propio del sensor. Este ruido está presente en todas las tomas, independientemente de la exposición y de la temperatura del sensor. Con el conjunto de tomas Bias se realiza un apilado promediando los valores de los píxeles homólogos de cada imagen, o bien tomando la mediana de dichos valores. Al hacer esto se crea un “**Master Bias**”.

1.4.2. Tomas Dark

Una de las correcciones que se pueden hacer a nuestras tomas científicas es la corrección por tomas Dark. Las tomas Dark se realizan con la misma configuración ISO que las tomas científicas y la misma exposición, pero “en la oscuridad”. Para ello, se tapa el objetivo de la cámara o del telescopio y se dispara una serie de tomas, usualmente más de diez. Las tomas Dark, entonces, no contienen “información”, pero contienen ruido térmico asociado con el sensor. Este ruido térmico es fuertemente dependiente de la temperatura y de la exposición. Así, para exposiciones menores a 30 s, pueden no ser necesarias estas tomas de calibración. Sin embargo, para exposiciones prolongadas, el sensor va aumentando su temperatura y el ruido térmico se hace más evidente. También la temperatura ambiente juega un rol fundamental en esta ecuación. Es por ello que las tomas Dark deben hacerse durante la misma sesión fotográfica, aunque eso quite tiempo de exposición para las tomas científicas.

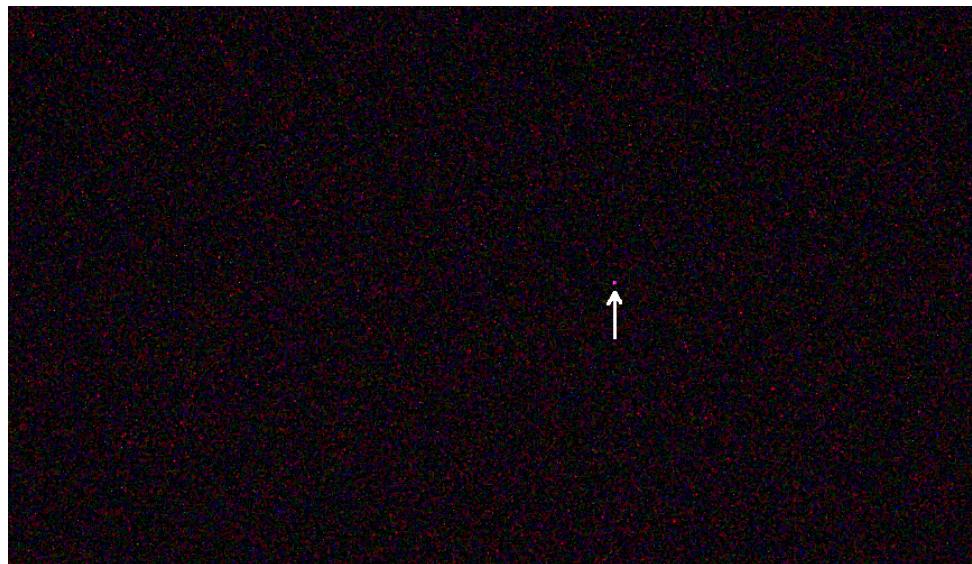


Figura 2: Recorte y ampliación de una toma Dark realizada con una DSLR Canon EOS 700D en un telescopio Maksutov-Cassegrain de 102 mm f/12,7. La imagen RAW está debayerizada y el contraste está levemente realzado. La flecha blanca muestra un defecto que no vuelve a aparecer en el resto de imágenes Dark de la serie y se puede deber, por ejemplo, al efecto de rayos cósmicos impactando en el sensor. Si este efecto apareciera en toda la serie de imágenes, entonces podríamos decir que es un defecto de lectura del sensor, conocido como “hot pixel”.

Con el conjunto de tomas Dark, usualmente se hace un apilado promediando los valores de los pixeles homólogos de cada imagen, o bien tomando la mediana de dichos valores. Sin embargo, como se decía en la sección 1.4.1, estas tomas también tendrán ruido electrónico y de lectura. Para quitar esta contribución, se le puede restar a cada toma Dark un Master Bias. Este último proceso es especialmente importante si las tomas Dark no tienen el mismo tiempo de exposición que las tomas científicas. Finalmente se deben apilar las imágenes Dark corregidas utilizando el promedio o la mediana de los pixeles, obteniendo así una imagen que se llama **“Dark-current”**. Antes de restar la imagen Dark-current a las tomas científicas, se las debe escalar por un factor igual al cociente entre el tiempo de exposición de las tomas científicas y de las tomas Dark.

1.4.3. Tomas Flat

Los telescopios y las lentes comerciales usualmente no iluminan el sensor en forma homogénea. Habitualmente, la luz se distribuye en mayor cantidad en el centro del sensor que en los bordes. Este defecto en las imágenes se llama viñeteo. Más aún, las superficies ópticas de los telescopios y lentes suelen acumular polvo (y el sensor de la cámara también). A esto hay que agregarle que el propio sensor no tiene la misma respuesta en todos sus pixeles a un mismo estímulo lumínico. La Figura 3 muestra una toma flat individual con el contraste realzado para mostrar algunos defectos que se pueden presentar en una imagen. A diferencia de los ruidos térmico y de lectura, que eran contribuciones aditivas a las tomas científicas, estos defectos son de carácter multiplicativo. Para corregir estos defectos, se debe dividir a las tomas científicas por un **“Master Flat”**.

Para fabricar un Master Flat, se deben sacar varias tomas apuntando el objetivo a una superficie uniformemente iluminada por luz blanca, sin llegar a saturar el sensor. Una manera

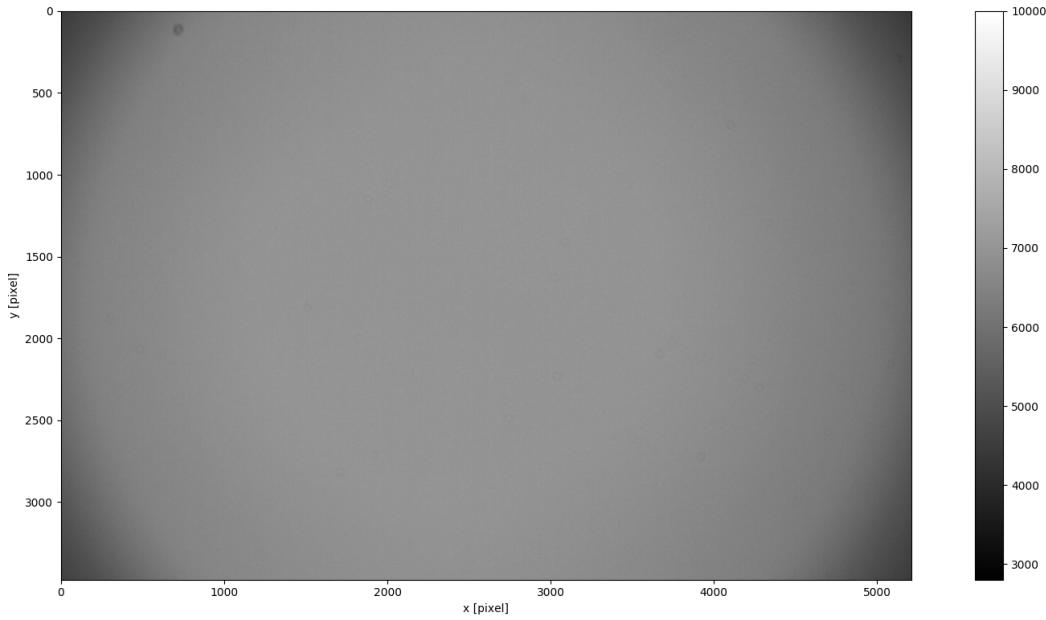


Figura 3: Toma Flat realizada con una DSLR Canon EOS 700D en un telescopio Maksutov-Cassegrain de 102 mm f/12,7. Los números en los ejes indican coordenadas en píxeles. La barra lateral indica los niveles de brillo del histograma. Se muestra solo el canal verde de la imagen. Se observa en las esquinas de la imagen el efecto de viñeteo debido a una iluminación no homogénea del sensor. También arriba a la izquierda de la imagen se ve una mancha circular que se debe a un resto de polvo en el tren óptico del instrumento.

simple y moderna de hacerlo es apuntar el telescopio a un monitor LCD (sin remover la cámara de su posición en el tubo). Las cámaras DSLR comerciales tienen un modo AV en el que la cámara calcula el tiempo de exposición automáticamente. Este modo usualmente proporciona Flats muy adecuados. La configuración ISO en el caso de las tomas Flats debe ser menor o igual a la de las tomas científicas.

Una vez que se tienen las tomas Flat, se les debe quitar el ruido de lectura y térmico restándole los Master Bias y Dark-current (escalado con el tiempo de exposición de los Flats). Luego se las apila utilizando la mediana, siendo el resultado obtenido el Master Flat. Antes de dividir las tomas científicas por el Master Flat, se le debe realizar un escalado o normalización, ya que es posible que los niveles de iluminación sean diferentes para cada imagen. Una opción posible sería darle peso a cada imagen Flat usando su mediana. Se promedian todas las imágenes Flats utilizando esos pesos, y se divide esta imagen resultante por el promedio de todos sus píxeles. La imagen Master Flat resultante está, entonces, reducida y normalizada.

Las correcciones explicadas en toda esta sección 3 se pueden condensar en la fórmula (1).

$$\text{Científicas}_{reducidas} = \frac{\text{Científicas}_{raw} - \text{Master Bias} - \text{Dark current}_{escalada}}{\text{Master Flat}_{reducida, \text{normalizada}}} \quad (1)$$

Esta fórmula de reducción de imágenes científicas que se utiliza aquí fue tomada del curso “Introduction to Astronomical Image Analysis” de Matthew Craig, Juan Cabanela y Linda Winkler [8].

Una vez que la serie de imágenes científicas fueron reducidas, se puede proceder a realizar las mediciones fotométricas. Sin embargo, si se busca realizar una medición precisa de un objeto muy débil, se puede aumentar aún más la SNR. Esto se logra realizando un apilado

(en inglés *stacking*) de un conjunto de tomas científicas reducidas de la misma región del cielo. Para este apilado existen diferentes algoritmos. El más sencillo consiste en promediar los pixeles homólogos de cada imagen, como se hacía con las tomas de calibración. Otros algoritmos consisten en realizar ese promedio, pero descartando valores extremos o reemplazándolos por valores que se encuentren dentro de ciertos percentiles.

Como último detalle, puede que la serie de imágenes no estén perfectamente alineadas, por ejemplo, porque el seguimiento del telescopio utilizado no es perfecto. Es por ello que muchas veces se hace necesario hacer una alineación, llevando todas las imágenes de la serie a un mismo sistema de coordenadas mediante traslaciones, rotaciones y, en casos más raros en que se usen distintas lentes, cambios de tamaño. Este proceso se llama “**Registro de imágenes**” [9].

2. Metodología

Para desarrollar el programa PyReduc se ha utilizado el lenguaje de programación Python 3. Este lenguaje de alto nivel, interpretado y orientado a objetos, posee una gran variedad de librerías científicas. Entre ellas, la librería NumPy es fundamental para trabajar con vectores, matrices y arreglos N-dimensionales. Integrado dentro de la gran librería SciPy, Matplotlib permite graficar *al vuelo* funciones, datos o imágenes de mapas de bits. Por otro lado, la librería Astropy es un esfuerzo comunitario por desarrollar todo un ecosistema de paquetes astronómicos que sean interoperables en lenguaje Python [10]. Entre ellos, el módulo `astropy.io.fits` permite manejar ficheros FITS. Está basado en el módulo PyFITS desarrollado por el Space Telescope Science Institute. El proyecto PyFITS se unió a Astropy como parte de estos esfuerzos de desarrollar una gran librería–ecosistema de paquetes astronómicos.

Dado que PyReduc está pensado para realizar reducciones que luego se utilizarán para fotometrías, el programa está pensado para procesar imágenes FITS monocromáticas. En el caso de imágenes RAW DSLR, será necesario, antes de utilizar PyReduc, realizar el debayerizado y extraer un canal con algún software externo. Una posibilidad sería utilizar el script de M. Emre Aydin `cr2fits` [11], que permite extraer un solo canal de una imagen RAW y la convierte en FITS, transcribiendo todos los metadatos necesarios.

Las imágenes FITS deben estar en un directorio llamado “`~/pyreduc/FITS/`”, donde “`~`” representa el directorio `HOME`. Los prefijos de los archivos deben seguir las siguientes reglas:

- FLATS: “`flat*`”
- DARKS: “`dark*`”
- BIAS: “`bias*`”

donde “`*`” significa “cualquier cosa a partir de aquí”. El prefijo de las tomas científicas se ingresa por teclado.

Por ejemplo:

En la carpeta “`~/pyreduc/FITS/`” se tienen los archivos:

- `bias001.fit, bias002.fit, bias003.fit bias004.fit,...`
- `darkA.fit, darkB.fit, darkC.fit,...`
- `flat_0001.fit, ..., flat_0005.fit, ..., flat_0011.fit`
- `RCnc_001.fit, RCnc_002.fit, RCnc_003.fit, ...`

entonces las tomas de calibración tienen el nombre correcto. Cuando el programa pida el prefijo de las tomas científicas, se deberá ingresar por teclado:

`RCnc`

Una vez que todos los archivos FITS de tomas científicas, Flats, Dark y Bias están dispuestas en una carpeta FITS dentro del directorio `~/pyreduc/FITS/`, PyReduc las copia, utilizando la librería `shutil`, a un nuevo directorio llamado `~/pyreduc/procesado/`, en el que se realizará el proceso de reducción.

Utilizando la librería `glob`, se arman 4 objetos de tipo `lista` de Python 3, que contendrán las tomas científicas, dark, bias y flat. Luego, empleando la librería `astropy.io.fits` se leen las cabeceras de estos archivos y se imprimen por pantalla los nombres de archivos junto con la resolución en pixeles de cada imagen y el tiempo de exposición.

La cantidad de archivos de cada lista se guarda en una variable distinta. Además se guardan en sendas variables el tiempo de exposición de cada tipo de toma. En este punto, hay que destacar que en PyReduc suponemos que todas las tomas de un mismo tipo tienen el mismo tiempo de exposición, por lo que puede que alguna parte del proceso no se cumpla si esta suposición no es verdadera.

Una vez hecho esto, comienza el proceso de reducción propiamente dicho. Este proceso está basado en un trabajo de Ricardo Gil-Hutton [5]. Este tutorial estaba basado en la librería `PyFits`, pero en PyReduc se ha reemplazado por `astropy.io.fits`.

Para realizar el procesado de Bias se creó una función llamada `mediana_calib`, cuyos argumentos son la lista¹ de imágenes a procesar y la cantidad de estas imágenes. La función creada ejecuta el procedimiento descripto a continuación:

1. Primero se arma un arreglo tridimensional en donde se ponen todas las imágenes Bias secuencialmente, de manera que el elemento a_{ijk} del arreglo es el pixel de coordenadas (j, k) de la i -ésima imagen Bias.
2. Luego se ordenan los pixeles de la secuencia de imágenes de menor a mayor a lo largo del primer eje. Para clarificar esto, se toman los pixeles de coordenadas fijas (j, k) de cada imagen de la secuencia y se los ordena de mayor a menor en el arreglo tridimensional. Así, la matriz que se obtendría de fijar el índice $i = 0$ estaría formada por los pixeles de menor valor de todas las imágenes². Este ordenamiento se realiza con la función `sort` de la librería NumPy.
3. Finalmente se calcula la mediana de todos los pixeles de coordenadas (i, j) fijas, pero descartando los pixeles de valor más alto. La razón de este descarte es eliminar posibles defectos como el que se ve en la Figura 2. Este cálculo de la mediana se realiza con la función `median` de la librería NumPy.

La imagen así obtenida constituye el Master Bias, el cual se guarda en memoria, en un array de NumPy de nombre `stbias`. A continuación, se sigue con el procesado de tomas Dark. Lo primero que se hace es restarle a todas las tomas Dark el Master Bias, formando así tomas pre-Dark-current. Para este fin, se creó una función llamada `resta_master` cuyos argumentos son la lista de imágenes a procesar y la imagen Master que se les va a restar. Cabe destacar que esta

¹Aquí *lista* se usa en el sentido del objeto Python de tipo lista.

²A diferencia del lenguaje FORTRAN, en el lenguaje Python los índices de los arreglos comienzan en 0.

función sobreescribe las imágenes de la lista que se le pasa como argumento, reemplazándolas por las nuevas imágenes pre-Dark-current.

Una vez que se tienen las imágenes pre-Dark-current, se utiliza la función `mediana_calib` para apilar estas tomas utilizando la mediana y descartando los valores de los píxeles más altos. La imagen así obtenida constituye la toma Dark-current buscada y se guarda en memoria en el arreglo `stdark`

Para realizar la calibración utilizando la fórmula (1), primero se crea una imagen que es la suma de la Master Dark y de la Dark-current escalado con el tiempo de exposición de las tomas científicas. Esta imagen, que en el programa se guarda en memoria en el arreglo de NumPy `master_stack`, obedece a la siguiente expresión:

$$\text{master_stack} = \text{Master Bias} + \frac{\text{exp}_{\text{dark}}}{\text{exp}_{\text{lights}}} \text{ Dark-current} \quad (2)$$

en donde exp_{dark} es el tiempo de exposición de las tomas dark y $\text{exp}_{\text{lights}}$ es el tiempo de exposición de las tomas científicas. Aquí es importante mencionar que en este programa estamos suponiendo que todas las tomas Dark tienen el mismo tiempo de exposición entre sí, y que lo mismo ocurre con las tomas científicas.

Una vez construida la toma de calibración `master_stack`, se la resta a todas las tomas científicas, recurriendo nuevamente a la función `resta_master`.

A continuación, se calibran todas las tomas Flat con una nueva imagen `master_stack`, construida utilizando la expresión (2), pero reemplazando el tiempo de exposición de las tomas científicas por el de las tomas Flat.

Para apilar las tomas Flat ya calibradas, el proceso es ligeramente diferente al de los Dark-current y los Bias, ya que estas tomas usan iluminación artificial. Como es posible que los niveles de iluminación de cada toma sean diferentes, se le da peso a cada imagen con su mediana. Se crea un arreglo tridimensional de imágenes Flat tal como se hacía para los Bias y los Dark, pero con dicho peso. Luego se calcula una nueva imagen promediando los píxeles homólogos de cada imagen y despreciando los valores más altos, pero se normaliza el resultado con la suma de las medianas de cada imagen. Así se obtiene una imagen Master Flat de buena relación señal/ruido, que se puede usar para hacer la corrección de las imágenes científicas. Esto se realiza dividiendo por el Master Flat y reescalando con su media para no modificar el valor medio de cada imagen.

De esta manera, la calibración basada en la fórmula de reducción (1) queda completada.

Antes de proceder al apilado de las imágenes, es necesario alinearlas. Para este proceso se ha escrito un módulo llamado `registrado` que es en realidad un script que utiliza las herramientas de `astroalign` para alinear las imágenes previamente calibradas. Astroalign es un paquete que alinea dos imágenes astronómicas buscando asterismos de tres puntos (tres estrellas) en común entre las dos imágenes, y realizando una transformación afín entre ellas. El autor del paquete `astroalign` es Martin Beroiz [12]. Dado que el proceso de alineación o registro de imágenes está fuera del alcance de este informe, solo se mencionará que la función `registra_lista`, dentro del módulo registrado, recibe como argumento la lista de imágenes a registrar y sobreescribe las imágenes de esa lista con nuevas versiones a las que se le aplicaron las transformaciones necesarias para un posterior apilado exitoso.

Una vez finalizado el alineado, el programa está listo para realizar el apilado de las tomas científicas. Para ello, se escribió un módulo llamado `apilado`, que tiene dos opciones a elegir por el usuario:

1. Apilado realizando el promedio de todos los píxeles homólogos de cada imagen.

2. Apilado con descarte de valores extremos (“pixel rejection”).

Para ambas opciones, se construye un arreglo tridimensional con todas las tomas científicas reducidas, igual que se hacía para las tomas de calibración. Luego, según la opción elegida por el usuario, este arreglo se pasa como argumento a una de las dos funciones escritas para el apilado.

Para la primera opción, la función `no_rejection` simplemente toma todos los pixeles homólogos de todas las tomas científicas ya calibradas y alineadas y realiza el promedio. El resultado es una nueva matriz de nombre de variable `stack` que se devuelve a la subrutina principal.

La segunda opción, que fue escrita desde cero para PyReduc, es algo más compleja. La idea del algoritmo contenido en la función `pixel_rejection` es tomar los pixeles homólogos de cada imagen y promediarlos, pero rechazando aquellos que se aparten más de dos desviaciones estándar del promedio. El algoritmo se muestra en el Apéndice C.

Finalizado el apilado, la imagen resultante se guarda en el archivo `stacking.fit`. Luego, el programa da la opción de visualizar dicha imagen. Para este fin, se ha escrito el módulo `visualizacion.py` que utiliza la librería `matplotlib` para representar la imagen en pantalla. En este punto es necesario mencionar que toda imagen se puede caracterizar con un histograma que de cuenta de la frecuencia con la que aparecen los diferentes valores de pixeles en la imagen. Dado que las imágenes FIT resultantes habitualmente tienen una cantidad de niveles de brillo muy extensa, para poder apreciar los detalles de la imagen se hace necesario retocar el histograma de la imagen de alguna manera. Para ello, el módulo de visualización consta de tres opciones:

1. La opción manual, que le muestra al usuario el histograma para que este elija un valor máximo y un mínimo entre los cuales recortar. Así, el usuario ajusta el histograma entre estos dos niveles de brillo. El usuario también puede escoger si al mostrarse la imagen se usa una interpolación para suavizarla o no.
2. El recorte automático del histograma entre los percentiles 2 y 98. Para este recorte automático se utiliza la función `exposure.rescale_intensity` de la librería `skimage`.
3. Una ecualización automática del histograma.

Para la última opción, se ha escrito para PyReduc un último módulo, `ecualizado.py` que es una implementación en lenguaje Python de un método muy difundido de ecualización de histogramas para el procesamiento digital de imágenes. La implementación se hizo a partir del algoritmo explicado en el libro “Image Processing in C” de Dwayne Phillips [4]. El método de ecualización *mapea* un histograma que presenta picos notorios en otro más *aplanado* o *ecualizado*. Aplanar el histograma hace que los pixeles oscuros aparezcan a la vista más oscuros y que los pixeles claros aparezcan a la vista más claros. En el Apéndice E se explica un poco más del método de ecualizado y se lista el código implementado.

Cabe destacar que las tres opciones de visualización trabajan en memoria RAM, por lo que ninguna de ellas altera realmente los datos de la imagen. En la ventana de visualización, el usuario puede encontrar un botón para guardar la imagen que está visualizando en un archivo con algún formato de mapa de bits tradicional, como el formato PNG. Al cerrar la ventana de visualización, las modificaciones al histograma se pierden.

Vale la pena mencionar, que este módulo de visualización se puede ejecutar independientemente del resto del programa PyReduc, siempre y cuando la imagen que se quiera visualizar tenga formato FITS y nombre de archivo `stacking.fit`.

3. Resultados

Para probar el programa PyReduc se utilizó una serie de imágenes de la nebulosa Messier 8 (M8), más conocida como Nebulosa Laguna. La secuencia de imágenes se realizó con una cámara Canon EOS 700D a foco primario en un telescopio Maksutov-Cassegrain de 102 mm de diámetro y relación focal f/12,7. Como se mostró en la Figura 3, este dispositivo presenta un viñeteado significativo, que vuelve imprescindibles las tomas de calibración Flat. Se emplearon 9 tomas científicas de 30 s de exposición a ISO 3200³. Para la calibración se realizaron 20 tomas Bias de 0,00025 s de exposición, 10 Dark de 30 s de exposición y 15 Flat de 0,25 s de exposición. Previamente a la utilización de PyReduc, se extrajo de todas las imágenes RAW el canal verde y se creó así una nueva secuencia de imágenes FITS, utilizando el programa Siril. De aquí en más siempre se trabajó con el canal verde de todas las imágenes.

En la Figura 4 se muestra una toma científica aún sin procesar de M8, tal como sale por la ventana de visualización empleada por PyReduc, con la opción de recorte automático del histograma entre los percentiles 2 y 98. La nebulosa se encuentra parcialmente enmascarada en el ruido. Obsérvese por ejemplo la región de píxeles de coordenadas (1500, 2000): se divisa allí una estrella y apenas detrás, la nebulosa se confunde con el fondo de la imagen.

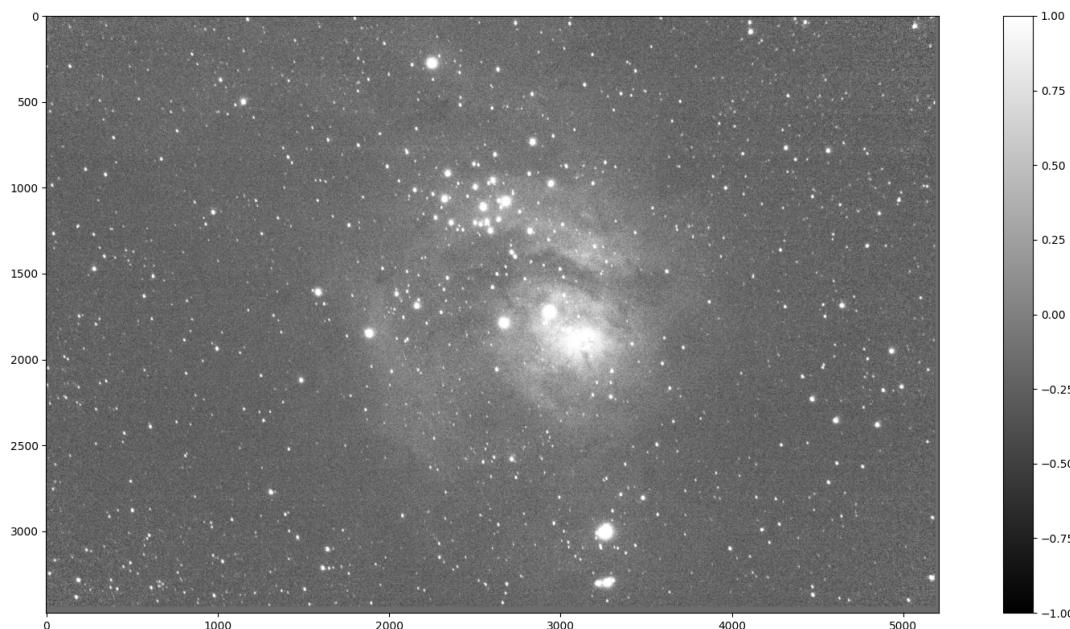


Figura 4: Toma científica de la Nebulosa Laguna (M8) sin procesar, con el histograma recortado entre los percentiles 2 y 98. Los números en los ejes indican coordenadas en píxeles. La barra lateral indica los niveles de brillo del histograma.

Luego de ejecutar el programa completo, con apilado por promedio simple de píxeles en las tomas ya calibradas y alineadas, se obtuvo la imagen que se muestra en la Figura 5. Obsérvese nuevamente en la región de píxeles de coordenadas (1500, 2000): alrededor de la estrella que se observaba antes “aislada”, ha aparecido una región oscura que antes apenas se apreciaba. La nebulosa se halla mucho más definida.

³Este valor de ISO es bastante elevado para realizar fotometría, ya que se aleja mucho de la ganancia de $1e^-/ADU$ del sensor, pero es útil para probar las funciones de PyReduc, por la cantidad de información que es capaz de muestrear

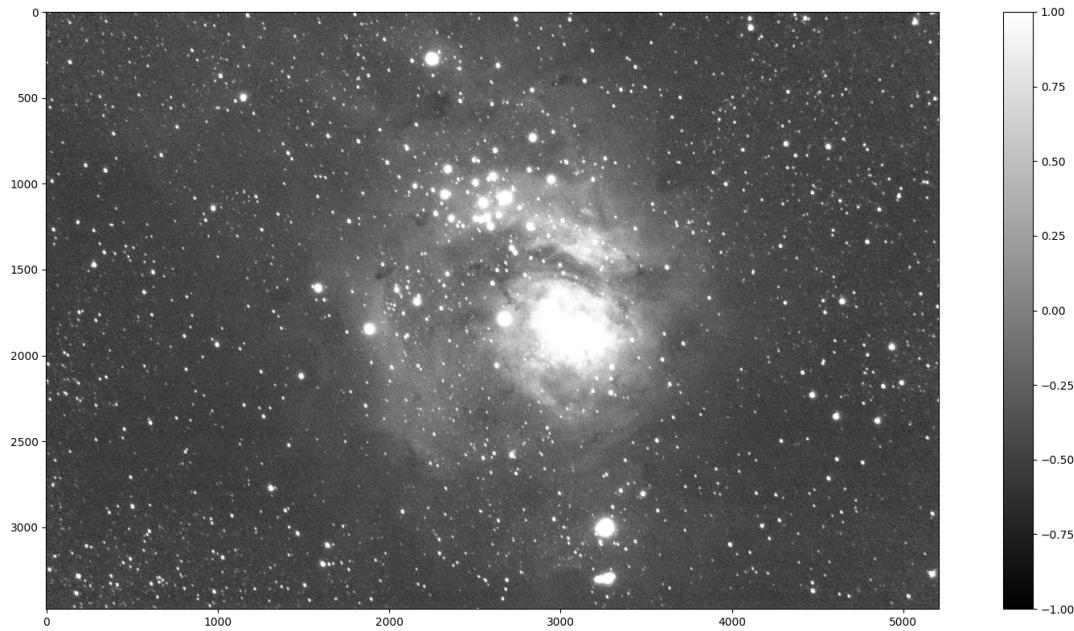


Figura 5: Imagen apilada de la Nebulosa Laguna (M8), con el histograma recortado entre los percentiles 2 y 98. Los números en los ejes indican coordenadas en píxeles. La barra lateral indica los niveles de brillo del histograma. Esta imagen está compuesta por 9 tomas científicas, 20 tomas Bias, 10 Dark y 15 Flat.

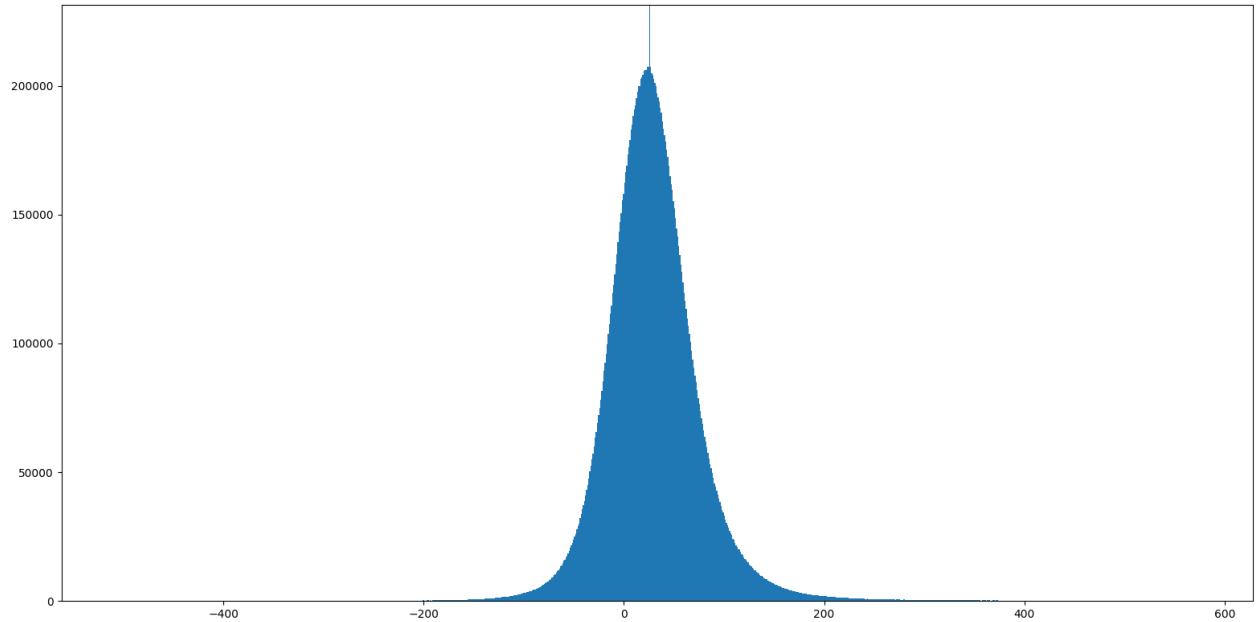


Figura 6: Histograma de la imagen sin procesar de la Figura 4. La región de interés está ampliada para notar detalles de la distribución.

A los fines de comparar en forma más cuantitativa las dos imágenes, la Figura 6 muestra el histograma de la imagen sin procesar, mientras que la Figura 7 muestra el histograma de la imagen final apilada. Se observa que en el histograma de la imagen sin apilar, la distribución tiene una dispersión mayor que en el histograma de la imagen final. Esta mayor dispersión en la

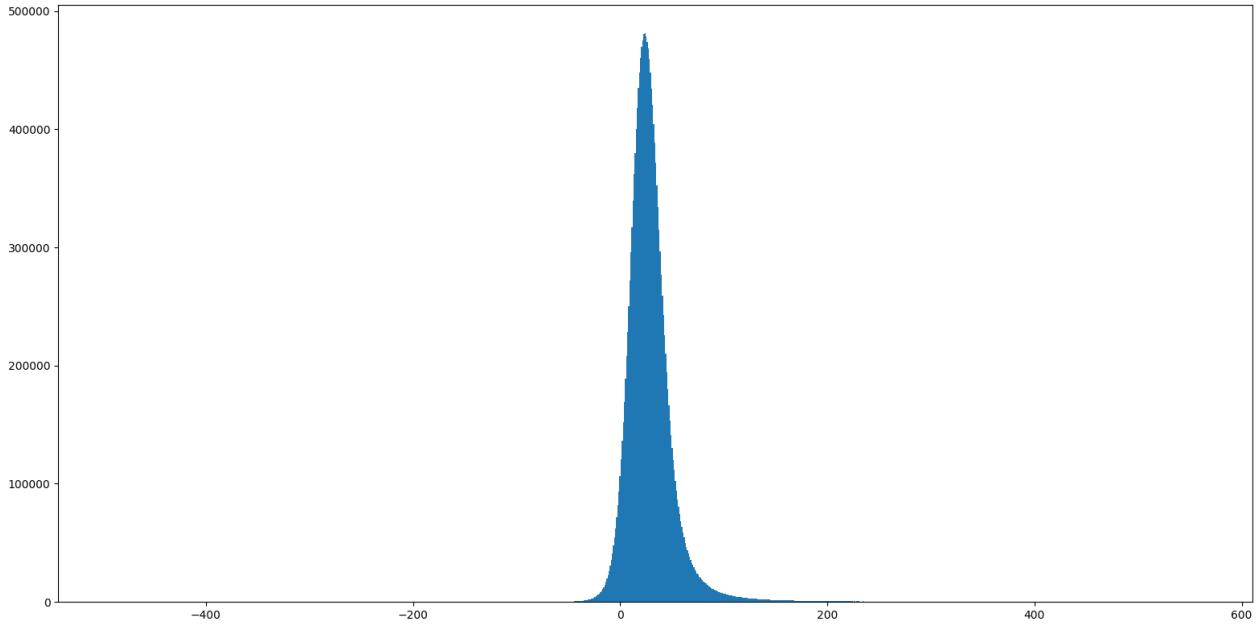


Figura 7: Histograma de la imagen final procesada de la Figura 5. La region de interés está ampliada para notar detalles de la distribución.

imagen sin procesar, visible en el ancho del pico y en su altura, es consistente con una presencia de mayor ruido en la imagen.

Una forma posible de probar la calidad de todo el proceso es medir la relación señal/ruido (SNR). Como se ha explicado en la sección , la reducción de imágenes debería incrementar la SNR, y también lo debería hacer el apilado. Para verificar que esto efectivamente se cumple con el procesado realizado con PyReduc, se ha realizado el siguiente procedimiento:

1. Se seleccionó un área cuadrada arbitraria de la imagen, de $50 \times 50 = 2500$ píxeles que mostrara el fondo del cielo (sin objetos astronómicos visibles). La esquina superior izquierda de dicha área estaba ubicada coordinadas (400, 400).
2. Se midió para el área elegida el valor promedio μ_{bg} y su desviación estándar σ_{bg} .
3. Se eligió una estrella arbitraria, no saturada, en la imagen⁴, y utilizando un cuadrado de $50 \times 50 = 2500$ píxeles se midió el valor promedio μ_s . La resta $\mu_s - \mu_{bg}$ representa la amplitud de la señal, eliminando el ruido, mientras que la desviación estándar del fondo del cielo representa el ruido.
4. Se calculó la relación señal/ruido en decibeles como

$$\text{SNR} = 10 \log \left(\frac{\mu_s - \mu_{bg}}{\sigma_{bg}} \right)$$

Aunque hay diferentes definiciones para la SNR, en este trabajo se eligió esta definición, algo diferente de la de la bibliografía [13, 14], por su practicidad a la hora de medir, sin necesidad de tomas de referencia o de calcular parámetros de la cámara.

⁴En un trabajo previo, fuera del alcance de este trabajo, se determinó que la cámara utilizada para estas pruebas perdía linealidad a las 15815 ADU. Estrellas en la imagen con valores ADU mayores que ese, estarán saturadas.

En la Figura 8 se muestra una gráfica de las mediciones de SNR realizadas con el proceso antes descripto, incrementando la cantidad de tomas científicas a apilar de a una por vez. También se incluyen las mediciones de una toma sin reducir, y de una sola toma reducida con Bias, Dark y Flats. Se observa que, efectivamente, una toma reducida tiene una mayor SNR que una toma sin reducir. Además se observa que a medida que se apilan más tomas, la SNR aumenta. Comprobamos así que el apilado, con el conjunto de imágenes de prueba aquí utilizado, mejora la SNR. Queda pendiente comprobar que esta mejora se sigue cumpliendo al apilar un mayor número de tomas científicas, digamos entre 50 y 100 tomas.

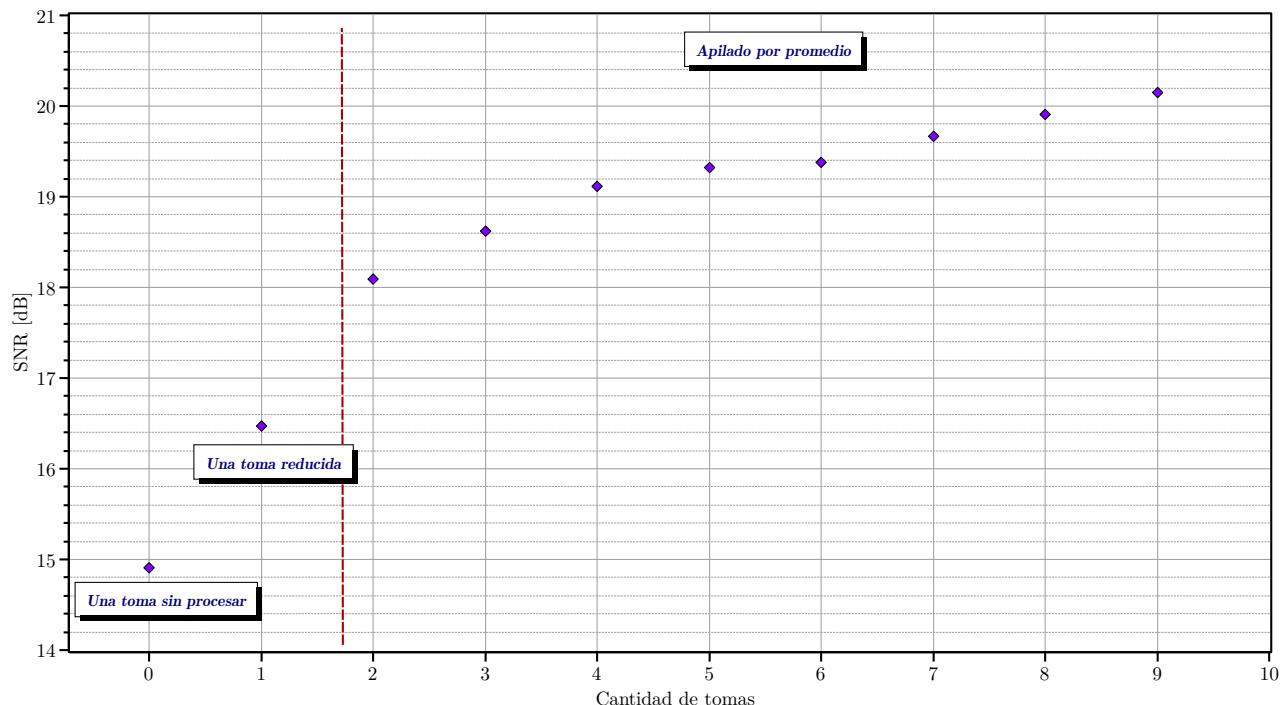


Figura 8: Relación señal/ruido (SNR) en función de la cantidad de tomas científicas apiladas. Se nota una mejora de la SNR a medida que se van agregando más tomas científicas. El valor de cero en las absisas indica una sola toma sin reducir, mientras que el valor uno indica una sola toma reducida por Bias, Dark y Flats.

Bibliografía

- [1] American Association of Variable Star Observers: “*Manual de Observación DSLR de AAVSO*”, Versión 1.1 en Español, 2015. En Internet: <https://www.aavso.org/dslr-observing-manual>
- [2] Free Software Foundation: “*Qué es el software libre?*”, Versión 1.153, 2017. En Internet: <https://www.gnu.org/philosophy/free-sw.es.html>
- [3] Eric W. Greisen: “*FITS: a remarkable achievement in information exchange*”, 2003. En: Heck A. (eds) Information Handling in Astronomy - Historical Vistas. Astrophysics and Space Science Library, vol 285. Springer, Dordrecht. https://doi.org/10.1007/0-306-48080-8_5

- [4] Dwayne Phillips: “*Image Processing in C*”, 2nd Ed, 2000. En Internet: <http://homepages.inf.ed.ac.uk/rbf/BOOKS/PHILLIPS/cips2ed.pdf>
- [5] Ricardo Gil-Hutton: “*Cómo hacer una reducción básica de imágenes FITS con Python*”, Noviembre 2016. En Internet: <https://freeshell.de/~rgh/arch/python/python-red-basica.pdf>
- [6] The FITS Support Office: “*What is FITS?*”, NASA. Última revisión, Diciembre 2017. En Internet: <https://fits.gsfc.nasa.gov/>
- [7] FITS Working Group, International Astronomical Union: “*Definition of the Flexible Image Transport System (FITS)*”, Version 4.0, 2016. En Internet: https://fits.gsfc.nasa.gov/standard40/fits_standard40aa-1e.pdf
- [8] Matthew Craig, Juan Cabanela, Linda Winkler: “*Introduction to Astronomical Image Analysis*”, 2014. En Internet: <https://image-analysis.readthedocs.io>
- [9] Lisa G. Brown: “*Image Registration Techniques*”, ACM Computmg Surveys, Vol 24, No. 4, 1992. <https://doi.org/10.1145/146370.146374>
- [10] Astropy Collaboration: “*The Astropy Project: Building an Open-science Project and Status of the v2.0 Core Package*”, The Astronomical Journal, Volume 156, Issue 3, article id. 123, 19 pp., 2018. <https://doi.org/10.3847/1538-3881/aabc4f>
- [11] M. Emre Aydin: “*cr2fits v2*”, Version v2.1.0, 2017, May 21, Zenodo. <http://doi.org/10.5281/zenodo.581722>
- [12] Martin Beroiz: “*toros-astro/astroalign*”, Version v1.0.3, 2018, May 18. Zenodo. <http://doi.org/10.5281/zenodo.1249048>
- [13] Roger L. Easton Jr.: “*Fundamentals of Digital Image Processing*”, Noviembre de 2010, Chester F. Carlson Center for Imaging Science, Rochester Institute of Technology. https://www.cis.rit.edu/class/simg361/Notes_11222010.pdf
- [14] W. Romanishin: “*An Introduction to Astronomical Photometry Using CCDs*”, Octubre de 2006, University of Oklahoma, En Internet: http://www.physics.csbsju.edu/370/photometry/manuals/OU.edu_CCD_photometry_wrccd06.pdf

A. Listado del código principal de PyReduc

En esta sección se lista el código contenido en el archivo `pyreduc.py`, módulo principal de PyReduc.

```

1  #!/usr/bin/python3
2
3  """
4  PyReduc
5  Programa de reducción de imágenes FITS
6  Autor: Carlos Mauricio Silva
7  Versión: 0.2.5
8
9  Licencia GNU GENERAL PUBLIC LICENSE
10 Leer archivo LICENSE que se distribuye con este programa.
11
12 Parte del código de calibración con Darks/Bias/Flats está basado en el
13   Tutorial "Cómo hacer una reducción básica de imágenes FITS con
14   Python" de Ricardo Gil-Hutton. Este tutorial estaba basado en la
15   librería PyFits, que ya no se usa y ha sido reemplazada por astropy
16   .io.fits.
17 La fórmula de reducción de imágenes científicas que se utiliza aquí
18   fue tomada del curso "Introduction to Astronomical Image Analysis"
19   de Matthew Craig, Juan Cabanella & Linda Winkler.
20 """
21
22 import numpy as np
23 from astropy.io import fits as ft
24 import glob
25 import os, shutil
26 from pathlib import Path
27
28 import apilado
29 import registrado
30 import visualizacion
31
32 # Presentación
33 print("-----")
34 print("PyReduc")
35 print("\nPrograma de reducción de imágenes FITS")
36 print("Autor: Carlos Mauricio Silva")
37 print("Versión: 0.2.0")
38 print("-----")
39 print("\nLas imágenes FITS deben estar en un directorio llamado")
40 print(''''pyreduc/FITS/'', dentro del directorio home.'''')
41 print("Los prefijos de los archivos deben seguir las siguientes reglas
42   :")
43 print('''' FLATS: "flat*"\n DARKS: "dark*"\n BIAS: "bias*"\n donde "*"
44   significa "cualquier cosa". El prefijo de los LIGHTS se ingresa por
45   teclado\n'''')
```

```

37
38 # Averiguo el nombre del directorio home
39 home = str(Path.home())
40
41 ##### FUNCIONES #####
42 #####
43 #####
44
45
46 def copia_de_imagenes():
47     # Hago una copia de todos los archivos para no sobreescribir los
48     # FITS
49     # En la carpeta ~/pyreduc/procesado/
50     home = str(Path.home())
51     os.chdir(home+"/pyreduc/FITS")
52     origen = os.getcwd()
53     destino = home+"/pyreduc/procesado/"
54     ignorar_pat = shutil.ignore_patterns('*.*')
55     if os.path.exists(destino): # Si el directorio destino existe, lo
56         elimino con todo su contenido
57         print("Antes de empezar a trabajar, se borrará el directorio
58             ~/pyreduc/procesado/")
59         input("Para cancelar la operación presione Ctrl+C. Para
60             continuar presione Enter")
61         shutil.rmtree(destino)
62         print("Aguarde mientras se crea una copia de sus imágenes\n")
63     try:                      # Y ahora lo vuelvo a crear copiando
64         todas las imagenes allí
65         arbol = shutil.copytree(origen, destino, ignore=
66             ignorar_pat)
67         print('Todas las imágenes se han copiado a', arbol)
68     except:
69         print('Error en la copia')
70     else:
71         print("Aguarde mientras se crea una copia de sus imágenes\n")
72     try:
73         arbol = shutil.copytree(origen, destino, ignore=
74             ignorar_pat)
75         print('Todas las imágenes se han copiado a', arbol)
76     except:
77         print('Error en la copia')
78
79
80 def mediana_calib(listaimg, numimg):
81     # Genero una matriz 3D de ceros
82     cubo_nuevo=np.zeros((numimg,ft.getval(listaimg[0], 'naxis2'),ft.
83         getval(listaimg[0], 'naxis1')),dtype=float)
84     # Copio las tomas de calibración a la matriz cúbica
85     nro=0
86     for ii in listaimg:

```

```

79     ff=ft.open(ii)
80     img=ff[0].data
81     hdr=ff[0].header
82     ff.close()
83     cubo_nuevo[nro,:,:,:]=np.copy(img)
84     nro+=1

85
86     # Ordeno los pixeles de menor a mayor a lo largo del primer eje.
87     cubo_nuevo=np.sort(cubo_nuevo, axis=0)

88
89     # Creo una nueva imagen con el valor de la mediana, despreciando
90     # el valor más alto de cada pixel.
91     return np.median(cubo_nuevo[0:numbias-1], axis=0)

92
93
94 def resta_master(lista,stacked_img):
95     for ii in lista:
96         ff=ft.open(ii)
97         img=ff[0].data
98         hdr=ff[0].header
99         ff.close()
100        img=img-stacked_img
101        hdr.add_comment("Procesado por DARK y BIAS con PyReduc")
102        ft.writeto(ii,img,header=hdr,overwrite=True) # overwrite=True
103        va a sobreescibir cada archivo.

104 #####
105 #### FIN FUNCIONES #####
106 #####
107 #####
108
109 copia_de_imagenes() # Llamo a la función que me backupeará las imá
110     genes
111
112 # Voy al directorio donde están las imágenes copiadas
113 os.chdir(home+"/pyreduc/procesado")
114
115 # Pido al usuario el prefijo de los archivos light
116 print("\nComenzando a trabajar con sus imágenes en el directorio", os.
117    .getcwd())
118 prefijo=input("\nIntroduzca el prefijo de los archivos lights (tomas
119     científicas): ")
120
121 # Construyo listas de tomas dark, bias, flat y lights
122 lista_dark=glob.glob("dark*.fit")
123 lista_bias=glob.glob("bias*.fit")
124 lista_flat=glob.glob("flat*.fit")
125 lista_lights=glob.glob(prefijo+"*.fit")

```

```

124 print("\n")
125 print("Archivos DARK:\n")
126 for ii in lista_dark:
127     print("{:}: {:}x{:} EXPTIME={:} s".format(ii,ft.getval(ii,'naxis2'
128                                         ),ft.getval(ii,'naxis1'),ft.getval(ii,'exptime')))
129
130 print("\n")
131 print("Archivos BIAS:\n")
132 for ii in lista_bias:
133     print("{:}: {:}x{:} EXPTIME={:} s".format(ii,ft.getval(ii,'naxis2'
134                                         ),ft.getval(ii,'naxis1'),ft.getval(ii,'exptime')))
135
136 print("\n")
137 print("Archivos FLAT:\n")
138 for ii in lista_flat:
139     print("{:}: {:}x{:} EXPTIME={:} s".format(ii,ft.getval(ii,'naxis2'
140                                         ),ft.getval(ii,'naxis1'),ft.getval(ii,'exptime')))
141
142 print("\n")
143 print("Archivos LIGHTS (tomas científicas):\n")
144 for ii in lista_lights:
145     print("{:}: {:}x{:} EXPTIME={:} s".format(ii,ft.getval(ii,'naxis2'
146                                         ),ft.getval(ii,'naxis1'),ft.getval(ii,'exptime')))
147
148 print("\nTenga en cuenta que para la calibración se toma el tiempo de
149 exposición de su primer toma científica.")
150 print("PyReduc no dará los mejores resultados si sus tomas científicas
151 tienen diferentes tiempos de exposición.")
152
153 continuar=input("Presione una tecla para continuar")
154
155 print("\nAl final del proceso se realizará un apilado de sus imágenes
156 calibradas y alineadas. Este se puede realizar a través de un
157 promedio simple de las imágenes, o con un promedio que rechace
158 valores extremos (Pixel Rejection). ¿Qué desea hacer?")
159 print("1. Apilar promediando las imágenes")
160 print("2. Aplicar Pixel Rejection (POCO EFICIENTE. NO RECOMENDADO.)")
161 print("3. No apilar")
162 rechazar_pixel = input("Ingrese 1, 2 o 3: ")
163 while(rechazar_pixel not in ["1", "2", "3"]):
164     print("Opción inválida")
165     rechazar_pixel = input("Ingrese 1, 2 o 3: ")
166
167
168 # Guardo la cantidad de archivos de cada lista en una variable
169 # distinta:
170 numflat=len(lista_flat)
171 numbias=len(lista_bias)
172 numdark=len(lista_dark)
173 numlights=len(lista_lights)

```

```

164
165 # Voy a obtener el tiempo de exposición de los lights, flats y darks
166 # Voy a suponer que todas las tomas de un mismo tipo tienen la misma
167 # exposición.
168 exp_lights=ft.getval(lista_lights[0], 'exptime')
169 exp_flat=ft.getval(lista_flat[0], 'exptime')
170 exp_dark=ft.getval(lista_dark[0], 'exptime')

171 # Proceso de BIAS. Voy a armar una matriz cúbica de los bias.
172 print("\nProcesando BIAS. Por favor, aguarde...")
173
174 stbias = mediana_calib(lista_bias, numbias)
175
176
177 # Proceso de DARK.
178 print("\nProcesando DARK. Por favor, aguarde...")
179
180 # Primero, voy a restar un master-bias a los dark, para formar una
181 # lista de pre-DARK-current.
182 print("\nSustrayendo un master-bias a los dark. Aguarde...")
183 resta_master(lista_dark, stbias)
184
185 # Ahora realizo el apilado para generar la imagen Dark-current
186 stdark = mediana_calib(lista_dark, numdark)
187
188
189 # Ahora voy a crear una imagen que es la suma de DARK-current escalado
190 # y BIAS
191 # para corregir los lights
192 master_stack = stbias + stdark/exp_dark*exp_lights
193
194 # Resto la nueva imagen a los lights
195 resta_master(lista_lights, master_stack)
196
197 # para corregir los flats
198 master_stack = stbias + stdark/exp_dark*exp_flat
199
200 # Resto la nueva imagen a los flats
201 resta_master(lista_flat, master_stack)
202
203
204 # Proceso de FLATS. Voy a armar una matriz cúbica de los flats.
205 # Nótese que los flats ya fueron corregidos con bias y dark-current.
206 print("\nProcesando FLATS. Por favor, aguarde...")
207 # Genero una matriz 3D de ceros
208 cubo_flat=np.zeros((numflat,ft.getval(lista_bias[0], 'naxis2'),ft.
209     getval(lista_bias[0], 'naxis1'))),dtype=float)
210
211 # Copio los FLATS a la matriz cúbica

```

```

210 nro=0
211 sum=0
212 for ii in lista_flat:
213     ff=ft.open(ii)
214     img=ff[0].data
215     hdr=ff[0].header
216     ff.close()
217     med=np.median(img)
218     sum+=med
219     cubo_flat[nro,:,:]=np.copy(img)*med
220     nro+=1
221
222 # Creo una nueva imagen a partir de los valores medios, despreciando
223 # los valores más altos,
224 # pero normalizo con la suma de las medianas de cada imagen.
225 stflat=np.mean(cubo_flat[0:numflat-1],axis=0)/sum
226 mflat=np.mean(stflat)
227
228 # Divido las imágenes lights por el master-flat resultante.
229 for ii in lista_lights:
230     ff=ft.open(ii)
231     img=ff[0].data
232     hdr=ff[0].header
233     ff.close()
234     img=img/stflat*mflat
235     hdr.add_comment("Procesado por FLATS con PyReduc")
236     ft.writeto(ii,img,header=hdr,overwrite=True)
237
238 # Alineación de imágenes:
239
240 registrado.registra_lista(lista_lights)
241
242 salir="1"
243
244
245 # Stacking o apilado de imágenes:
246 lista_lights=glob.glob(prefijo+"*.fit") # Reconstruyo la lista de
247 # lights
248 # Si el usuario pidió apilar, se apila, y si no, se sale del programa.
249 if(rechazar_pixel=="1" or rechazar_pixel=="2"):
250     apilado.stack(lista_lights, rechazar_pixel)
251 else:
252     salir="2"
253
254
255 while(salir=="1"):
256     print("¿Qué desea hacer?")
257     print("1: Ver la imagen resultante")
258     print("2: Salir")

```

```

258     salir=input("Ingrese 1 o 2: ")
259     if(salir=="1"):
260         # Visualización por pantalla de la imagen:
261         visualizacion.visual("stacking.fit")
262     elif(salir=="2"):
263         break
264     else:
265         print("Opción no válida")
266         salir="1"
267
268
269     print("Gracias por utilizar PyReduc")
270
271 import gc
272 # Saco la basura y se la lleva el camión recolector
273 gc.collect()

```

B. Listado del código de alineación con Astroalign

En esta sección se lista el script utilizado para alinear las tomas calibradas utilizando el software Astroalign. Dicho script está contenido en el módulo `registrado.py`.

```

1 # Este módulo es en realidad un script que utiliza las herramientas de
2 # astroalign para alinear las imágenes previamente calibradas con
3 # flats, darks y bias en PyReduc.
4 #
5 # Astroalign es un simple paquete que alinea dos imágenes astronómicas
6 # buscando asterismos de tres puntos (tres estrellas) en común entre
7 # las dos imágenes, y realizando una transformación afín entre ellas
8 # . El autor del paquete astroalign es Martin Beroiz.
9 # https://github.com/toros-astro/astroalign
10
11
12
13 def imprimir_info(p, ii):
14     # Esta función imprime por pantalla la info de la transformación
15     # que se aplicará.
16     # Aunque es innecesaria, sirve para que el usuario sepa que la má-
17     # quina está haciendo algo.
18     print("\nAlineando imagen {}".format(ii))
19     print("Rotación: {:.2f} grados".format(p.rotation * 180.0 / np.pi)
20          )
21     print("Factor de escala: {:.2f}".format(p.scale))

```

```

19     print("Traslación: (x, y) = ({:.2f}, {:.2f})".format(*p.
20             translation))
21
22 def registra_lista(lista):
23     cantidad=len(lista)
24
25     # La primera imagen de la lista será la toma de referencia.
26     print("\nComenzando la alineación.")
27     print("\nLa toma de referencia es {}".format(lista[0]))
28     blanco=ft.open(lista[0])
29     img_blanco=blanco[0].data
30     hdr_blanco=blanco[0].header
31     blanco.close()
32     del(lista[0]) # Quito la imagen de referencia del listado
33     for ii in lista:
34         ff=ft.open(ii)
35         img_torcida=ff[0].data
36         hdr_torcida=ff[0].header
37         ff.close()
38         p, (pos_img, pos_img_rot) = astroalign.find_transform(
39             img_torcida, img_blanco)
40         imprimir_info(p, ii)
41         img_aligned = astroalign.register(img_torcida, img_blanco)
42         hdr_torcida.add_comment("Registrado con Astroalign y PyReduc")
43         ft.writeto(ii, img_aligned, header=hdr_torcida, overwrite=True)
44
45     print("\nRegistrado realizado con éxito")

```

C. Listado del código de apilado de PyReduc

En esta sección se explica el código utilizado por PyReduc para apilar las tomas científicas ya calibradas y alineadas. Este módulo, contenido en el archivo `apilado.py`, consta de tres funciones: la función principal, `stacking`, recibe como argumento la lista de imágenes a apilar y las organiza en un arreglo tridimensional. Además, recibe como argumento una variable llamada `rechazar`, que vale 1 si el usuario decidió realizar un promedio simple de las imágenes, o vale 2, si el usuario decidió que se apile con un algoritmo de descarte de valores extremos. En función de esa decisión, la función `stacking` llama a una de las dos funciones restantes: `no_rejection` para un promedio simple de las imágenes o `pixel_rejection` de lo contrario. La función `no_rejection` toma como argumento el arreglo tridimensional de imágenes a apilar y devuelve, el promedio de todas esas imágenes en una matriz de NumPy de nombre de variable `stack`.

La función `pixel_rejection` tiene un desarrollo más complejo y fue escrita para PyReduc. Toma como argumento el arreglo tridimensional `cubo` y un valor `m` preestablecido en la función `stacking`. En el código actual, `m` vale 2, y significa que se ignorarán para el promedio todos los valores de pixeles que se alejen más de 2 desviaciones estándar del valor medio. El algoritmo recorre el arreglo tridimensional de imágenes, en el que el elemento a_{ijk} es el pixel de coordenadas

(j, k) de la i -ésima imagen. Para cada par (j, k) fijo, se calcula el promedio

$$\overline{a_{jk}} = \sum_{i=0}^{N-1} a_{ijk}$$

y la desviación estándar del promedio σ_{jk} . Luego se calculan los valores extremos que se van a aceptar:

$$maskmin = \overline{a_{jk}} - m\sigma_{jk}$$

$$maskmax = \overline{a_{jk}} + m\sigma_{jk}$$

Como se ha dicho, m en este caso toma el valor 2. Finalmente, en la línea 20 del algoritmo, se usa un *array enmascarado* de NumPy (Masked Array) para promediar los valores de a_{ijk} (con j y k fijos) que se encuentran en el intervalo $[maskmin, maskmax]$ ignorando el resto de los valores.

Al igual que en la otra opción de apilado, la función `pixel_rejection` devuelve una matriz llamada `stack` que contiene la imagen promediada. Antes de finalizar, de regreso en la función `stacking`, se guarda la imagen resultante en el archivo `stacking.fit`.

El módulo recién descripto se lista a continuación:

```

1 # Módulo de apilado para PyReduc.
2 # Autor: Carlos Mauricio Silva
3 # Recibe una lista de lights y realiza un apilado basado en el
4 # promedio de los pixeles.
5 # Hay un pixel rejection pero lleva MUCHO tiempo, así que no se
6 # recomienda.
7
8 import numpy as np
9 from astropy.io import fits as ft
10 import numpy.ma as ma
11
12 def pixel_rejection(cubo, m):
13     numimages, absi, orde = cubo.shape
14     stack = np.zeros((absi, orde))
15     data = np.zeros(numimages)
16     print(absi, orde)
17     for i in range(absi):
18         for j in range(orde):
19             data[:] = cubo[:, i, j]
20             maskmin = np.mean(data) - np.std(data) * m
21             maskmax = np.mean(data) + np.std(data) * m
22             stack[i, j] = ma.masked_outside(data, maskmin, maskmax).
23                           mean()
24             print(i, j)
25     return np.array(stack)
26
27 def no_rejection(cubo):
28     # Creo una nueva imagen con el valor de la media.
29     stack = np.mean(cubo, axis=0)
30     return np.array(stack)

```

```

28
29
30 def stacking(lista, rechazar):
31     cantidad=len(lista)
32     print("\nComenzando el apilado...")
33     # Genero una matriz 3D de ceros
34     cubo=np.zeros((cantidad,ft.getval(lista[0], 'naxis2'),ft.getval(
35         lista[0], 'naxis1')),dtype=float)
36     # Copio los lights a la matriz 3D
37     nro=0
38     for ii in lista:
39         ff=ft.open(ii)
40         img=ff[0].data
41         hdr=ff[0].header
42         ff.close()
43         cubo[nro,:,:]=np.copy(img)
44         nro+=1
45     #
46     if(rechazar=="1"):
47         stack = no_rejection(cubo)
48     else:
49         stack = pixel_rejection(cubo, m=2)
50     ft.writeto("stacking.fit",stack,header=hdr,overwrite=True)
51     print("Apilado realizado con éxito. La salida se ha guardado en el
52         archivo stacking.fit")

```

D. Listado del código de visualización de imágenes

En esta sección se lista el código utilizado para mostrar por pantalla la imagen final obtenida con el apilado. Este módulo, contenido en el archivo `visualizacion.py`, tiene tres opciones de visualización: con ecualizado automático, con recorte manual del histograma y con recorte automático entre los percentiles 2 y 98.

```

1 # Script de visualización de imágenes FITS.
2 # Primero se muestra un histograma de la imagen y se le pide
3 # al usuario que acote el rango de visualización máximo y mínimo
4 # a partir del histograma. Otra opción posible es que el programa
5 # ajuste automáticamente el histograma a través de una ecualización.
6 # Por último, también se proporciona la opción de reescalar el
7 # histograma entre los percentiles 2 y 98, solo con fines de explorar
8 # la librría skimage. Las otras opciones de esa librería no
9 # proporcionaron buenos resultados.
10
11 import numpy as np
12 from astropy.io import fits as ft
13 import matplotlib
14 import matplotlib.pyplot as plt

```



```
50     # Si quiero probar la ecualización de histograma de skimage,
51     # en lugar de la «casera»
52     # exposure.equalize_hist(imagen)
53     #
54     plt.imshow(img_rescale, cmap='gray', interpolation='bilinear')
55     plt.colorbar()
56     plt.show()

57
58 def caso_invalido(imagen):
59     print("Opción inválida")

60 print(n)
61 tabla = {"1": caso_manual, "2": caso_ecualizador, "3":
62     caso_avanzado}
63 opcion = tabla.get(n, caso_invalido)
64 opcion(imagen)

65
66
67 def visual(imagen):
68     image_data = ft.getdata(imagen)
69     print("\nEstadísticas de la imagen:")
70     print('Min:', np.min(image_data))
71     print('Máx:', np.max(image_data))
72     print('Promedio:', np.mean(image_data))
73     print('Desvío Estándar:', np.std(image_data))
74     #
75     print("\nPara visualizar la imagen correctamente, se puede
76         realizar un clipping (recorte manual) personalizado del
77             histograma, elegir el ecualizado automático o un stretching
78                 automático entre los percentiles 2 y 98. ¿Qué desea hacer?")
79     print("1. Recorte Manual del histograma")
80     print("2. Ecualización automática del histograma")
81     print("3. Recorte automático entre los percentiles 2 y 98")
82     n = input("Ingrese 1, 2 o 3: ")
83     modo_histograma(n, image_data)

84
85 # Las siguientes líneas sirven para probar el Script sin necesidad de
86 # ejecutar el main.
87 if __name__ == '__main__':
88     import os
89     from pathlib import Path
90
91     home = str(Path.home())
92     os.chdir(home+"/pyreduc/procesado")
93     visual("stacking.fit")
```

E. Método de ecualización del histograma

En esta sección, por último, se explica brevemente el método de ecualización del histograma de una imagen y se lista el código del módulo `ecualizado.py` para implementar la ecualización del histograma de la imagen final obtenida con PyReduc.

Como se explicó anteriormente, el método de ecualización *mapea* un histograma que presenta picos notorios en otro más *aplanado* o *ecualizado*. La operación de ecualización se puede representar con una función f que, al operar sobre una imagen c , la transforma en otra imagen b con un histograma aplanado:

$$b(x, y) = f[c(x, y)] \quad (3)$$

La ecuación (4) muestra la función densidad de probabilidad de hallar un pixel con el valor a en la imagen.

$$p_1(a) = \frac{1}{A} H_1(a) \quad (4)$$

donde A es el área de la imagen en pixeles y $H_1(a)$ es la distribución que da el histograma de la imagen. Sumando todas las funciones densidad de probabilidad hasta el valor a , obtenemos la función densidad acumulativa (cdf):

$$P_1(a) = \frac{1}{A} \sum_{i=0}^a H_1(i) \quad (5)$$

Si D es el número de niveles de gris en la imagen ecualizada b , se deberá cumplir

$$D = \frac{1}{p(a)}$$

para cada valor de pixel a en la imagen b . Esa es la razón por la que decimos que al ecualizar, aplanamos el histograma: todos los valores de pixeles aparecen la misma cantidad de veces. Para obtener la función de ecualización $f(a)$, multiplicamos, entonces, el número de niveles de grises D de la imagen b por la función densidad acumulativa, obteniendo así

$$f(a) = D \frac{1}{A} \sum_{i=0}^a H_c(i) \quad (6)$$

Cuando la imagen tiene un histograma *aplanado*, el histograma acumulativo asociado, dado por su cdf tiene forma de función lineal, monótonamente creciente. Una imagen de este tipo tiene un máximo contraste, lo cual quiere decir que puede exhibir los más sutiles cambios a lo largo de todo su rango de grises. Se dice en este caso que la información⁵ de la imagen fue maximizada [13].

El algoritmo implementado en Python 3 para el ecualizado automático de la imagen final se lista a continuación:

```

1 # Módulo de ecualización de histograma.
2 # Este módulo es una implementación en lenguaje Python
3 # del difundido método de ecualización de histogramas

```

⁵En el sentido de información de Shannon

```
4 # para el procesamiento digital de imágenes.  
5 #  
6 # La implementación se hizo a partir del algoritmo explicado en  
7 # "Image Processing in C" pp. 33-43, del autor Dwayne Phillips.  
8 # http://dwaynephillips.net/  
9 # Implementación realizada por Carlos Mauricio Silva  
10 # para Pyreduc.  
11  
12 import numpy as np  
13  
14 def histograma(imagen, absi, orde, grises):  
15     # Función que arma el histograma de la imagen.  
16     hist = np.zeros(grises)  
17     for i in range(absi):  
18         for j in range(orde):  
19             k = imagen[i,j]  
20             hist[k] = hist[k] + 1  
21     return np.array(hist)  
22  
23  
24 def histograma_acumulativo(hist):  
25     sum = 0  
26     hist_acum = np.zeros(len(hist))  
27     for gris in range(len(hist)):  
28         sum = sum + hist[gris]  
29         hist_acum[gris] = sum  
30     return np.array(hist_acum)  
31  
32  
33 def transformacion(imagen):  
34     imagen = imagen.astype(int) # convierto la imagen a enteros 32  
     bits con signo  
35     grises = np.max(imagen)-np.min(imagen) # cant. de niveles de gris  
     de la imagen  
36     absi, orde = imagen.shape    # Medidas de la imagen  
37     area = absi * orde  
38     coef = grises/area  
39     hist = histograma(imagen, absi, orde, grises)  
40     hist_acum = histograma_acumulativo(hist)  
41     salida = np.zeros((absi, orde))  
42     for i in range(absi):  
        for j in range(orde):  
            k = int(imagen[i,j])  
            salida[i,j]=coef*hist_acum[k]  
43     return np.array(salida)
```